On the Complexity of Bounded-Variable Queries

(Extended Abstract)

Moshe Y. Vardi*
Rice University

Abstract

It is known that the expression complexity and combined complexity of query evaluation in relational databases are in general exponentially higher than the data complexity of query evaluation. This gap in complexity is caused by the fact that evaluating a relational query may involve intermediate results that are exponentially larger than the input. In this paper we study the complexity of query evaluation with intermediate results of polynomial size. This bound is accomplished by restricting the queries to have a bounded number of individual variables. We show that, for bounded-variable queries, the gap between their expression and combined complexities, one one hand, and their data complexity, on the other hand, narrows and in some cases disappears. These results confirm the practice of trying to minimize the size of intermediate relations in database query evaluation, and suggest variable minimization as a query optimization methodology.

1 Introduction

One of the most fundamental issues in the theory of relational databases concerns the computational complexity of query evaluation. This issue was first tackled in [CM77] and it has remained an active research topic since then; see [Cha88] for a survey. It is now well established that there is a tradeoff between expressive power and computational complexity; the more expressive a query language, is the harder it is to evaluate queries in this language [CH82, Var82].

A taxonomy for measuring the computational complexity of query languages was developed in [Var82]. The taxonomy distinguishes between three cases. In the first case, we have a fixed query that is evaluated against different databases; in the second case, we evaluate different queries against a fixed database; and in the third case we evaluate different queries against different databases. In the first case, we measure the complexity as a function of the length of the data; we call this the *data complexity* of the query language under consideration. In the second case, we measure the complexity as a function of the length of the expression denoting the query; we call this the *expression complexity* of the query language under consideration. In the third case, we measure the complexity as a function of the combined length of the database and the expression denoting the query; we call this the *combined complexity* of the query language under consideration.

Using this taxonomy, a systematic study of the complexity of query evaluation for various query languages was undertaken in [Var82]. The main languages studied there are *first-order logic* (FO), *fixpoint logic* (FP), *existential second-order logic* (ESO), and *partial-fixpoint logic* (PFP).[1] This study revealed the phenomenon that, in general, there is an exponential gap between data complexity and expression complexity, while the latter essentially coincides with combined complexity. For example, while the data complexity of FP is PTIME-complete, the expression complexity and combined complexity of FP are both EXPTIME-complete.

What is the reason for this exponential gap between data complexity, on one hand, and expression complexity and combined complexity, on the other hand? As observed in [Cos83], evaluating a relational query may involve intermediate results that are exponentially larger than the input. For example, an intermediate result in the evaluation of a relational algebra expression e, e.g., a join of many relations, may involve a relation whose arity is linear in the length of e, which means that its size might be exponential in the length of e.

While in general intermediate results in query evalua-

*Work partly done at the IBM Almaden Research Center. Address: Dept. of Computer Science, MS 132, 6100 Main St., Rice University, Houston, TX 77005-1892, USA. Phone: 713-285-5977, 713-285-5930 (fax). E-mail: vardi@cs.rice.edu.

[1] To be precise, rather than PFP, the language studied in [Var82] is that of RQL queries [CH82], but this language was shown in [AV89] to have the same expressive power and complexity as PFP.

tion can be rather large, for many queries of practical interest this blow-up can be avoided. Consider, for example, a database that consists of the following relations: EMP(Emp,Dept), MGR(Dept,Mgr), SCY(Mgr,Scy), and SAL(Emp,Sal), containing information about employees and their departments, departments and their managers, managers and their secretaries, and employees and their salaries, respectively. Suppose that we want to evaluate the query "Find employees who earn less money than their manager's secretary". A naive approach would start by taking the cross product of EMP, MGR, SCY, SAL, and SAL, yielding a 10-ary relation, and then would follow by selecting and projecting appropriately. A better approach would be to take the natural join EMP with MGR and project on {Emp, Mgr} to obtain the binary relation EMP-MGR(Emp,Mgr); join the latter with with SCY and project on {Emp, Scy} to obtain EMP-SCY(Emp,Scy); and then join twice with SAL and select appropriately to get the answer. In this approach the largest arity of an intermediate relation is 4. This phenomenon is well-known. In fact, the fundamental reason that *acyclic* joins are easier to evaluate than *cyclic* joins [BFMY83, Yan81] is that they can be evaluated without large intermediate results.

In this paper we wish to study the complexity of evaluating queries with intermediate results of polynomial size. What is the syntax that captures the essence of this restriction? We propose a restriction that was recently shown to be extremely useful in the context of finite-model theory (e.g., [KV92, Hod93]). Consider a language $\mathcal{L}$ that includes a notation for individual variables (as is the case with FO, FP, ESO, and PFP). Normally, we assume an infinite supply $x_0, x_1, \ldots$ of individual variables. One can obtain a *bounded-variable* version $\mathcal{L}^k$ of the language $\mathcal{L}$ by restricting the individual variables to be among $x_1, \ldots, x_k$, for $k > 0$. The intuition is that, since each subexpression of a query in $\mathcal{L}^k$ has free variables among $x_1, \ldots, x_k$, each intermediate relation has arity of at most k, and is therefore of polynomial size.

Our investigation confirms this intuition by showing that, for bounded-variable query languages, their expression complexity and combined complexity shifts towards their data complexity. (Since data complexity deals with fixed queries, the bounded-variable restriction is essentially irrelevant to data complexity.) In many cases the gap completely closes, but in other cases it does not. For example, while the combined complexity of ESO is NEXPTIME-complete and the data complexity of ESO is NP-complete, the combined complexity of ESO^k is NP-complete. As another example, while the combined complexity of FP is EXPTIME-complete and the data complexity of FP is PTIME-complete, the best bound we have for the combined complexity of FP^k is NP$\cap$co-NP. These results confirm the practice of trying to minimize the size of intermediate relations in database query evaluation [Ull89], and suggest variable minimization as a query optimization methodology.

The latter bound is also of independent interest, because of an application to the problem for verifying finite-state programs. A finite-state program Π can be viewed as a relational database consisting of unary and binary relations. Verifying that the program Π satisfies its specification φ amounts to evaluating φ as a query against Π (cf. [CES86]). The specification language $L\mu$, also called the *propositional μ-calculus*, which has been widely studied in the literature (cf. [Koz83]), can be shown to be a fragment of FP^2. Thus, a bound on the combined complexity of FP^2 would also yield a bound on the complexity of verifying $L\mu$-properties.

The complexity of verifying $L\mu$-properties was studied in numerous papers [EL86, Cle90, SW91, Win91, Cle92, Cle93, And94]. The best known upper bound is NP$\cap$co-NP [EJS93] (see also [BVW94]). This bound is proven by reducing the problem to the emptiness problem for tree automata, which is known to be in NP [Eme85, VS85] (in fact, it is NP-complete [EJ88]). The reduction depends on the syntax of $L\mu$ and does not seem to apply to FP^2. Here we prove that the same bound applies to FP^k for any $k > 0$. Unlike the proof in [EJS93], our proof here is direct and relies on basic properties of least and greatest fixpoints.

2 Basic Definitions

2.1 Databases, Queries, and Complexity

We first recall some basic definitions from [CH82, Var82].

A *relational database* (or *database*, for short) is a tuple $B = (D, R_1, \ldots, R_\ell)$, where the *domain* $D \subseteq \mathbb{N}$ is a finite set of natural numbers and for each $1 \leq i \leq \ell$, we have $R_i \subseteq D^{a_i}$ for some $a_i \geq 0$. The number a_i is called the *arity* of the *relation* R_i, and B is said to be of arity $\mathbf{a} = (a_1, \ldots, a_k)$. We usually abbreviate the tuple $R_1, \ldots, R_k$ by $\mathbf{R}$ and write $B = (D, \mathbf{R})$. Throughout the paper, we assume some standard encoding for databases, e.g., the database $B = (\{3,5,7\}, \{\langle 3,5\rangle, \langle 5,7\rangle\})$ might be encoded by the string $(\{011, 101, 111\}, \{\langle 011, 101\rangle, \langle 101, 111\rangle\})$.

A *query* of arity $\mathbf{a} \to b$ is a partial function

$$Q : \{B \mid B \text{ is of arity } \mathbf{a}\} \to 2^{\mathbb{N}^b}$$

such that if $B = (D, \mathbf{R})$ and $Q(B)$ is defined then $Q(B) \subseteq D^b$. We are interested only in total queries in this paper. (Chandra and Harel [CH80] require that queries be computable and generic. These conditions are satisfied by the query languages studied in this paper, so we will not address this point any further.) A *query language* (or *language*, for short) is a set of expressions $\mathcal{L}$ such that with every expression e in

$\mathcal{L}$, we associate a query Q_e. We say that e *denotes* Q_e. We often blur the distinction between e and Q_e. Throughout this paper we assume some standard encoding for expressions.

Since queries are functions, it is more convenient to measure their complexity by considering the associated decision problem; that is, we measure the complexity of deciding if $\mathbf{t} \in Q(B)$, where $\mathbf{t}$ is a tuple, Q is a query, and B is a databases. As we mentioned in the introduction, we distinguish between the cases where the query is fixed, the database is fixed, or neither the query nor the database is fixed. The *answer set* of a query Q is the set[2]

$$Answer(Q) = \{(\mathbf{t}, B) \mid \mathbf{t} \in Q(B)\}.$$

The answer set of a database B with respect to a language $\mathcal{L}$ is the set

$$Answer_{\mathcal{L}}(B) = \{(\mathbf{t}, e) \mid e \in \mathcal{L} \text{ and } \mathbf{t} \in Q_e(B)\}.$$

Finally, the answer set of a language $\mathcal{L}$ is the set

$$Answer_{\mathcal{L}} = \{(\mathbf{t}, B, e) \mid e \in \mathcal{L} \text{ and } \mathbf{t} \in Q_e(B)\}.$$

We measure the complexity of query evaluation by measuring the complexity of answer sets. The *data complexity* of a language $\mathcal{L}$ is the complexity of the sets $Answer(Q_e)$ for queries e in $\mathcal{L}$, the *expression complexity* of $\mathcal{L}$ is the complexity of the sets $Answer_{\mathcal{L}}(B)$, and the *combined complexity* of $\mathcal{L}$ is the complexity of the set $Answer_{\mathcal{L}}$.

2.2 Query Languages

We study in this paper four query languages: *first-order logic* (FO), *existential second-order logic* (ESO), *fixpoint logic* (FP), and *partial-fixpoint logic* (PFP). Given a formula $\varphi(\mathbf{x})$ in any of these logics with free individual variables among $\mathbf{x} = (x_1, \ldots, x_m)$, the expression $(\mathbf{x})\varphi(\mathbf{x})$ denotes a query defined by

$$Q_{(\mathbf{x})\varphi(\mathbf{x})}(D, \mathbf{R}) = \{\mathbf{t} \in D^{|\mathbf{x}|} \mid (D, \mathbf{R}) \models \varphi(\mathbf{t})\}.$$

FO is essentially Codd's *relational calculus* [Cod72]. ESO is the logic consisting of formula of the form $(\exists \mathbf{S})\varphi(\mathbf{x}, \mathbf{S})$, where φ is first order. Here, φ can refer not only to the database relations but also to the existentially quantified relations in $\mathbf{S}$. ESO was first studied in [Fag74], where it was shown that ESO defined precisely the class of queries whose data complexity is NP.

FP is obtained by augmenting FO with the least-fixpoint operator [CH82]. Let $\varphi(\mathbf{x}, \mathbf{y}, S)$ be a formula with free individual variables among $\mathbf{x} = (x_1, \ldots, x_m)$ and $\mathbf{y}$ in which an m-ary relation symbol S occurs positively (i.e., in the scope of an even number of negations). Given an interpretation α of the variables in $\mathbf{y}$, the formula φ can be viewed as an operator from m-ary relations to m-ary relations:

$$\varphi(P) = \{\mathbf{t} \mid \varphi(\mathbf{t}, \alpha(\mathbf{y}), P) \text{ holds}\}.$$

Because S occurs positively in φ, the operator φ is monotone, i.e, if $P \subseteq Q$, then $\varphi(P) \subseteq \varphi(Q)$. Since the sequence

$$\emptyset \subseteq \varphi(\emptyset) \subseteq \varphi(\varphi(\emptyset)) \cdots$$

is increasing, it has a limit, denoted φ^{∞}, which is the least fixpoint of φ. The formula $\mu S(\mathbf{x}).\varphi(\mathbf{x}, \mathbf{y}, S)(\mathbf{z})$ (whose free variables are those in $\mathbf{y}$ and $\mathbf{z}$) refers to this least fixpoint. Formally we have that $\mu S(\mathbf{x}).\varphi(\mathbf{x}, \mathbf{y}, S)(\mathbf{t})$ holds if $\mathbf{t} \in \varphi^{\infty}$. We say that S is a *recursive* relation in $\mu S(\mathbf{x}).\varphi(\mathbf{x}, \mathbf{y}, S)(\mathbf{z})$.

It is convenient to refer directly to greatest fixpoints, which are the complements of least fixpoints. That is, if φ is as above, then the formula $\nu S(\mathbf{x}).\varphi(\mathbf{x}, \mathbf{y}, S)(\mathbf{z})$ refers to the greatest fixpoint of φ, which is the limit φ_{∞} of the decreasing sequence

$$D^m \supseteq \varphi(D^m) \supseteq \varphi(\varphi(D^m)) \cdots.$$

Formally we have that $\nu S(\mathbf{x}).\varphi(\mathbf{x}, \mathbf{y}, S)(\mathbf{t})$ holds if $\mathbf{t} \in \varphi_{\infty}$. Again, we say that S is a *recursive* relation in $\mu S(\mathbf{x}).\varphi(\mathbf{x}, \mathbf{y}, S)(\mathbf{z})$.

Note that FP allows nesting of least fixpoints, Boolean connectives, and quantifiers. For example, the sentence $\mu S(x).\varphi(x, S)(u)$, where $\varphi(x, S)$ is

$$\nu T(z).\forall y(E(z, y) \rightarrow (S(y) \vee (P(y) \wedge T(y))))(x)$$

says "there is no infinite E-path starting at u on which P fails infinitely often".[3] It is known that over ordered databases, FP expresses precisely all queries whose data complexity is in PTIME [Imm86, Var82].

PFP is obtained by augmenting FO with the partial-fixpoint operator [AV89]. Let $\varphi(\mathbf{x}, \mathbf{y}, R)$ be a formula with free individual variables among $\mathbf{x} = (x_1, \ldots, x_m)$ and $\mathbf{y}$ in which an m-ary relation symbol R occurs (not necessarily positively). We still can view φ as an operator as above, but now it is not necessarily a monotone operator. Thus, the sequence

$$\emptyset, \varphi(\emptyset), \varphi(\varphi(\emptyset)), \cdots$$

is not necessarily increasing and may not have a limit. If it has a limit, we denote this limit by φ^{∞}, otherwise we take φ^{∞} to denote the empty m-ary relation. We refer to φ^{∞} as the *partial* fixpoint of φ. The formula $\pi R(\mathbf{x}).\varphi(\mathbf{x}, \mathbf{y}, R)(\mathbf{z})$ refers to this partial

[2] Here and later on, $\mathbf{t}$ and B are understood to be of the right arity.

[3] An infinite E-path of a possibly finite graph $G = (V, E)$ is an infinite sequence $v_0, v_1, \ldots$ such that $\langle v_i, v_{i+1} \rangle \in E$ for all $i \geq 0$.

fixpoint. Formally we have that $\pi R(\mathbf{x}).\varphi(\mathbf{x},\mathbf{y},R)(\mathbf{t})$ holds if $\mathbf{t} \in \varphi^{\infty}$. It is known that over ordered databases, PFP expresses precisely all queries whose data complexity is in PSPACE [Var82, AV89].

Let $\mathcal{L}$ be any of the query language described above. Normally we assume an infinite supply $x_0, x_1, \ldots$ of individual variables. One can obtain a *bounded-variable* version $\mathcal{L}^k$ of the language $\mathcal{L}$ by restricting the individual variables to be among $x_1, \ldots, x_k$, for $k > 0$.

Suppose, for example, that we want to express in FO the fact that nodes x and y in a graph are connected by a path of length n, for $n > 0$. A naive approach would use $n+1$ variables. Let $\psi_n(x,y)$ be $\exists z_1, \ldots, z_{n-1}(E(x,z_1) \wedge \cdots \wedge E(z_{n-1},y)$, where E is the edge relation. By reusing variables, one can express $\psi_n(x,y)$ in FO^3. Define $\varphi_1(x,y)$ to be $E(x,y)$, and define $\varphi_{n+1}(x,y)$ to be $\exists z(E(x,z) \wedge \exists x(x = z \wedge \varphi_n(x,y)))$. It is easy to see that $\varphi_n(x,y)$ is equivalent to $\psi_n(x,y)$. FO^k corresponds to the fragment of relational algebra where the arity of every subexpression is bounded by k. See [IK89] for an investigation of the expressive power of bounded-variable FO.

Our goal in this paper is to study the complexity of evaluating queries in FO^k, FP^k, ESO^k, and PFP^k, for $k > 0$. The intuition is that in these languages every subexpression denotes a relation whose arity is at most k, and consequently its size it at most n^k (where n is the number of elements in the database). Thus, in bounded-variable languages the intermediate results in query evaluation are of polynomial size.

2.3 Complexity of Query Evaluation

The investigation in [CH82, Var82, Imm87] yielded a complexity classification for FO, ESO, FP, and PFP, per Table 1.

The data-complexity entry for FO refers to *uniform* AC^0. (For a precise discussion of the data complexity of FO, see [Imm89, BIS90].) All other entries indicate upper and lower bounds. For example, the data-complexity entry for FP indicate that for each FP-query φ, the answer set $Answer(\varphi)$ is in PTIME, and there is an FP-query φ_0 such that the answer set $Answer(\varphi_0)$ is PTIME-hard. Similarly, the expression-complexity entry for FP indicate that for each database B, the answer set $Answer_{FP}(B)$ is in EXPTIME, and there is a database B_0 such that the answer set $Answer_{FP}(B_0)$ is EXPTIME-hard.[4]

3 Combined Complexity

Since the data-complexity lower bounds involve fixed queries, these lower bounds apply also to the combined complexity of the bounded-variable version of the query languages under consideration, provided that the number of variables is above a certain threshold. That is, the combined complexity of a bounded-variable language $\mathcal{L}^k$ is bounded below by the data complexity of $\mathcal{L}$ for large enough k. Since intermediate results in bounded-variable queries are polynomially bounded, we would hope to avoid the exponential gap between data complexity, on one hand, and combined complexity, on the other hand. Thus, we would expect the combined complexity of $\mathcal{L}^k$ to be the same as the data complexity of $\mathcal{L}$. As Table 2, which summarizes our results, indicates, this is indeed the case for ESO and PFP, while the gaps between the combined complexity and the data complexity narrows for FO and FP.

For FO we do not get the expected collapse; the combined complexity of FO^k is higher than the data complexity of FO, since uniform AC_0 is properly contained in PTIME [Ajt83, FSS81]. For FP we can only prove a partial collapse; we have an upper bound of NP$\cap$co-NP. Improving this bound is a challenging problem.

We now describe these complexity bounds in detail.

3.1 FO^k

We first prove a polynomial-time upper bound. This bound is implicit in [Imm82, proof of Theorem B.5].

Proposition 3.1: *$Answer_{FO^k}$ is in PTIME for $k > 0$.*

Proof sketch: Let $(\mathbf{y})\varphi(\mathbf{y})$ be an FO^k-query. By definition, the variables of $\mathbf{y}$ are among $\mathbf{x} = (x_1, \ldots, x_k)$. We show how to compute the answer to $(\mathbf{x})\varphi(\mathbf{x})$ in polynomial time; clearly the answer to $(\mathbf{y})\varphi(\mathbf{y})$ can be computed then in polynomial time, since the answer to $(\mathbf{y})\varphi(\mathbf{y})$ can be computed from the answer to $(\mathbf{x})\varphi(\mathbf{x})$ by projecting out some columns and then permuting the remaining columns.

The basic idea is to view the subformulas of φ as subqueries. That is, if ψ is a subformula of φ, then before evaluating $(\mathbf{x})\varphi(\mathbf{x}))$ we evaluate $(\mathbf{x})\psi(\mathbf{x}))$. Thus, the evaluation is bottom-up and all intermediate results are k-ary relations. For example, to evaluate $(\mathbf{x})\varphi_1 \wedge \varphi_2(B)$ we take the intersection of $(\mathbf{x})\varphi_1(B)$ and $(\mathbf{x})\varphi_2(B)$. ∎

It follows from [Imm82, Theorem B.5] that $Answer_{FO^k}$ is PTIME-hard for all large enough k. We show here that this holds for all $k > 2$.

Proposition 3.2: *$Answer_{FO^k}$ is PTIME-hard for $k > 2$.*

Proof sketch: The proof is by reduction from Path Systems [Coo74]. Here the database consists of one ternary relation Q and two unary relations S and T. The set of *reachable* elements is defined by the following

[4] Actually, this lower bound holds for any *nontrivial* database, i.e., a database whose domain D that contains at least 2 elements and a nonempty k-ary relation P, $k \geq 1$, that is different from D^k.

Datalog program.

$$\begin{array}{lcl} P(x) & \leftarrow & S(x) \\ P(x) & \leftarrow & Q(x,y,z), P(y), P(z) \end{array}$$

The Path System query is to decide whether T contains any reachable elements.

Consider the FO^3-formula $\varphi(x)$:

$$S(x) \vee \exists y \exists z (Q(x,y,z) \wedge \forall x((x = y \vee x = z) \rightarrow P(x))).$$

Define $\varphi^1(x)$ to be $\varphi(z, P(x)/\mathbf{false})$, and $\varphi^n(x)$ to be $\varphi(z, P(x)/\varphi^{n-1}(x))$. Note that $\varphi^n(x)$ is of size $O(n)$ and it is in FO^3. Let ψ^n be $\exists x(T(x) \wedge \varphi^n(x))$. It is not hard to show that the Path System query holds in a database B with m elements iff ψ^m holds in B. ∎

3.2 FP^k

It is tempting to think that FP^k-queries ought to be evaluable in polynomial time. Because of the restriction on the number of variables, the arity of the recursive relations in formulas of FP^k is at most k. Thus, each fixpoint computation $\emptyset \subseteq \varphi(\emptyset) \subseteq \varphi(\varphi(\emptyset)) \cdots$ takes at most n^k iterations, and by Proposition 3.1, each iteration can be computed in polynomial time.

Unfortunately, this reasoning fails to take into account the possible nesting of fixpoints, and in particular, alternation of least and greatest fixpoints. Consider for example the FP-formula

$$\nu P(\mathbf{x}).\varphi(P, \mu Q(\mathbf{y}).\psi(Q, P, \nu R(\mathbf{z}).\theta(R, P, Q))).$$

How does one compute the greatest fixpoint νP? A straightforward approach would apply the following program:

$P \leftarrow D^k$
Repeat until convergence
$P \leftarrow \varphi(P, \mu Q(\mathbf{y}).\psi)$
Endrepeat

Each iteration, however, requires a computation of the least fixpoint μQ by the program:

$Q \leftarrow \emptyset$
Repeat until convergence
$Q \leftarrow \psi(P, Q, \nu R(\mathbf{z}).\theta)$
Endrepeat

Here each iteration requires the computation of the greatest fixpoint νR by the program:

$R \leftarrow D^k$
Repeat until convergence
$R \leftarrow \theta(P, Q, R)$
Endrepeat

Thus, the overall program consists of a triply nested loop, and instead of n^k iterations, we need n^{3k} iterations. In general, the number of required iterations is n^{kl}, where l is the depth of nesting of alternating fixpoint operations. This bound is exponential in the length of the expression denoting the query.[5]

One conceivable approach to deal with this difficulty is to apply what is known as Immerman's Theorem in order to eliminate the alternation of least and greatest fixpoints. It is shown in [Imm86] that one can express greatest fixpoints over finite structures in terms of least fixpoints. This seems to imply that alternation of least and greatest fixpoints can be collapsed to a single least fixpoint. Unfortunately, it is not clear to the author how to apply Immerman's proof technique to *nested* fixpoints, when the fixpoints are more than doubly nested. The same difficulty applies to the alternative proof of Immerman's Theorem in [Lei90]. In contrast, the proof technique in [GS86] does apply to alternating fixpoints and results in the collapse of the alternation hierarchy. This collapse, however, is achieved through an increase in the arity of the fixpoints; l-nested k-ary fixpoints are collapsed to a single kl-ary least fixpoint. Unfortunately, computing kl-ary fixpoints is exponential in l, which unlike k is not fixed. Thus, collapsing alternation does not seem to yield any computational benefit.

Our approach to evaluating fixpoint queries is based on approximating both least and greatest fixpoints from below, as opposed to the standard approach, where least fixpoints are computed from below and greatest fixpoints are computed from above. The key to our approach is the following two lemmas.

Lemma 3.3: *Let f be a monotone function on sets Then $a \in gfp(f)$ iff there exist a function $f' \sqsubseteq f$ and a set Q such that $a \in Q$ and $Q \subseteq f'(Q)$.*

Proof: By Tarski-Knaster's Theorem [Tar55], $\mathrm{gfp}(f) = \cup\{Q \mid Q \subseteq f(Q)\}$. If $a \in \mathrm{gfp}(f)$, then the claim is satisfied by taking $f' = f$ and $Q = gfp(f)$. On the other hand, if there exist a function $f' \sqsubseteq f$ and a set Q such that $a \in Q$ and $Q \subseteq f'(Q)$, then $Q \subseteq f(Q)$, so $Q \subseteq gfp(f)$ and $a \in gfp(f)$. ∎

Lemma 3.4: *Let f be a monotone function on finite sets. Then $a \in lfp(f)$ iff there exist monotone functions*

[5] It seems that this difficulty would also arise in formulas that contain nested μ's, e.g.,

$$\mu P(\mathbf{x}).\varphi(P, \mu Q(\mathbf{y}).\psi(Q, P, \mu R(\mathbf{z}).\theta(R, P, Q))).$$

Indeed a naive computation of nested μ's would be exponential in the depth of the nesting. This, however, can be avoided. When all recursive relations are least-fixpoint relations, all the least-fixpoint computations are monotonic. This monotonicity can be taken advantage of to evaluate the query in ln^k iterations instead of n^{kl} iterations.

$f_1 \sqsubseteq f_2 \cdots \sqsubseteq f$ *such that* $a \in \bigcup_{i=0}^{\infty} Q_i$, *where* $Q_0 = \emptyset$, *and* $Q_i = f_i(Q_{i-1})$.

Proof: By Tarski-Knaster's Theorem [Tar55], $\mathrm{lfp}(f) = \bigcup_{i=0}^{\infty} P_i$, where $P_0 = \emptyset$ and $P_i = f(P_{i-1})$, for $i > 0$. If $a \in \mathrm{gfp}(f)$, then the claim is satisfied by taking $f_1 = f_2 = \cdots = f$.

Suppose, on the other hand, that there exist monotone functions $f_1 \sqsubseteq f_2 \cdots \sqsubseteq f$ such that $a \in \bigcup_{i=0}^{\infty} Q_i$, where $Q_0 = \emptyset$, and $Q_i = f_i(Q_{i-1})$. We claim that the sequence $Q_0, Q_1, \ldots$ is increasing and $Q_i \subseteq P_i$, where $P_0 = \emptyset$, and $P_i = f(P_{i-1})$. Trivially, $Q_0 \subseteq P_0$. Suppose that $Q_{i-1} \subseteq Q_i$ and $Q_i \subseteq P_i$. Then $Q_i = f_i(Q_{i-1}) \subseteq f_i(Q_i) \subseteq f_{i+1}(Q_i) = Q_{i+1}$. Also, $Q_{i+1} = f_{i+1}(Q_i) \subseteq f(Q_i) \subseteq f(P_i) = P_{i+1}$. It follows that $a \in \bigcup_{i=0}^{\infty} Q_i \subseteq \bigcup_{i=0}^{\infty} P_i = \mathrm{lfp}(f)$. ∎

Theorem 3.5: *Answer*$_{FP^k}$ *is in NP∩co-NP.*

Proof sketch: Apply Lemmas 3.3 and 3.4 top-down. For example, to decide whether $\mathbf{t}$ is in

$$\nu P(\mathbf{x}).\varphi(P, \mu Q(\mathbf{y}).\psi(Q, P, \nu R(\mathbf{z}).\theta(R, P, Q)))(\mathbf{u})(B),$$

we guess a set P, and check that $\mathbf{t} \in P$ and $P \subseteq \varphi(P, Q)(\mathbf{x})(B)$, where Q under-approximates

$$\mu Q(\mathbf{y}).\psi(Q, P, \nu R(\mathbf{z}).\theta(R, P, Q))(\mathbf{u})(B).$$

To compute Q, we run the program

$i \leftarrow 0$
$Q_0 \leftarrow \emptyset$
Repeat until convergence
$Q_{i+1} \leftarrow \psi(P, Q_i, R_i)$
$i \leftarrow i + 1$
Endrepeat
$Q \leftarrow Q_i$

For each iteration here, we guess an under-approximation R_i of

$$\nu R(\mathbf{z}).\theta(R, P, Q_i)(\mathbf{u})(B).$$

That is, for the ith iteration we guess a set R_i such that $R_i \supseteq R_{i-1}$ and $R_i \subseteq \theta(P, Q_i, R_i)$.

Overall, the number of iterations here is reduced by this technique from n^{3k} to $3n^k$ at the cost of introducing nondeterminism. Generally, this technique reduces the number of iterations from n^{kl}, where l is the alternation depth, to ln^k. Since, by Proposition 3.1, each iteration can be performed in polynomial time, this yields an NP upper bound. The co-NP upper bound follows from the fact that $\mathbf{t} \notin (\mathbf{x})\varphi(\mathbf{x})(B)$ iff $\mathbf{t} \in (\mathbf{x})\neg\varphi(\mathbf{x})(B)$. ∎

The proof of Theorem 3.5 depends crucially on the monotonicity of the formulas in FP^k. It is known that FP is equivalent in expressive power to IFP (*inflationary fixpoint logic*) [GS86]. Our techniques do not seem to apply to IFP^k, the bounded-variable version of IFP, and the best known upper bound for the combined complexity of IFP^k is the polynomial space bound that follows from the combined complexity of PFP^k (see Section 3.4).

3.3 ESO^k

The source of the difficulty in evaluating ESO^k-queries is that the bound on the number of individual variables does not bound the arity of the relation variables. This is in contrast to FP, where the arity of the relation variables was bounded by the bound on the number of individual variables. Thus, the naive approach to evaluating an ESO^k-query $(\mathbf{x})(\exists \mathbf{S})\psi(\mathbf{x}, \mathbf{S})$, i.e., guess the existentially quantified relations and then check whether ψ holds, does not work, since the size of the quantified relations can be exponential in the length of ψ.

The key to overcome this difficulty is the observation that only a polynomial-size fragment of the quantified relation is used is evaluating ψ. Consider an atomic subformula $S_i(u_1, \ldots, u_l)$ of ψ. Even though it contains l individual variables, all these variables must be among $x_1, \ldots, x_k$. Thus, each reference to S_i defines a selection condition on S_i according to the pattern of equalities among the individual variables $u_1, \ldots, u_l$. Overall, ψ contains only a linear number of such atomic formulas. This is the key to the following lemma.

Lemma 3.6: *Every ESO^k-formula ψ is equivalent to an ESO^k-formula ψ', where the arity of the quantified relations is at most k and $|\psi'|$ is at most polynomially bigger than $|\psi|$.*

Proof sketch: Consider an atomic subformula $S_i(\mathbf{u})$ of ψ. Even though it contains $|\mathbf{u}|$ individual variables, all these variables must be among $x_1, \ldots, x_k$. The number of such atomic formulas is linear in $|\psi|$. For each such subformula, we introduce a new k-ary predicate symbol $S_i^{\mathbf{u}}$, we replace the second-order existential quantification $\exists \mathbf{S}$ by second-order existential quantification over the new k-ary predicate symbols, and we replace each atomic subformula $S_i(\mathbf{u})$ in φ by $S_i^{\mathbf{u}}(x_1, \ldots, x_k)$. For example, suppose that $k = 2$, S is 4-ary, and φ contains the atomic subformulas $S(x_1, x_1, x_2, x_2)$ and $S(x_1, x_2, x_1, x_2)$, then S is replaced by $S^{\langle x_1, x_1, x_2, x_2\rangle}$ and $S^{\langle x_1, x_2, x_1, x_2\rangle}$, and the S-atomic subformulas are replaced by $S^{\langle x_1, x_1, x_2, x_2\rangle}(x_1, x_2)$ and $S^{\langle x_1, x_2, x_1, x_2\rangle}(x_1, x_2)$, respectively.

The intuition is that $S_i^{\langle x_{i_1}, \ldots, x_{i_l}\rangle}$ is a "view" of S_i defined by the FO^k query $(x_1, \ldots, x_k)S_i(x_{i_1}, \ldots, x_{i_l})$. We now need to assert that all these views are consistent with each other. For example, suppose that $S^{\langle x_1, x_1, x_2, x_2\rangle}(a, a)$ holds. This represents the fact that $S(a, a, a, a)$ holds, which means that $S^{\langle x_1, x_2, x_1, x_2\rangle}(a, a)$

should hold. Thus, we need to assert

$$\forall x_1(S^{\langle x_1,x_1,x_2,x_2\rangle}(x_1,x_1) \leftrightarrow S^{\langle x_1,x_2,x_1,x_2\rangle}(x_1,x_1)).$$

More generally, let $u_1,\ldots,u_k$ and $v_1,\ldots,v_k$ and be k-sequences of (not necessarily distinct) variables among $x_1,\ldots,x_k$ such that the l-sequences $u_{i_1},\ldots,u_{i_l}$ and $v_{j_1},\ldots,v_{j_l}$ coincide. Then we need to assert

$$\forall x_1\cdots x_k(S_i^{\langle x_{i_1},\ldots,x_{i_l}\rangle}(u_1,\ldots,u_k) \leftrightarrow S_i^{\langle x_{j_1},\ldots,x_{j_l}\rangle}(v_1,\ldots,v_k)).$$

The number of such assertions is quadratic in $|\psi|$. Adding these assertions to the revised ψ yields the desired ψ'. ∎

Corollary 3.7: *$Answer_{ESO^k}$ is in NP.*

3.4 PFP^k

It is not hard to see that the straightforward approach to evaluating PFP-queries uses only polynomial space. The key observation is that, as in FP, the arity of the relation variables is bounded by the bound on the number of individual variables.

Theorem 3.8: *$Answer_{PFP^k}$ is in PSPACE.*

4 Expression Complexity

The upper bounds that we have proven for the combined complexity of bounded-variable queries clearly apply also to their expression complexity. Since, however, the database is now fixed, the lower bounds of data complexity may not necessarily apply here. It is conceivable that the expression complexity might be lower than the combined complexity.

As Table 3, which summarizes our results, indicates, this is indeed the case for FO, while for ESO and PFP the previous lower bound still apply (though they need to be reproved). For FP^k, we only have the same upper bound as for combined complexity.

We now describe these complexity bounds in detail.

4.1 FO^k

The key to the lower expression complexity of FO^k is the observation that if the database is fixed, then there are only finitely many k-ary relations. Thus, an FO^k-query can be viewed as an "algebraic" expression over a finite "algebra". How hard it is to evaluate such expressions? This question was answered in a classical paper about the complexity of *parenthesis languages* [Lyn77].

Recall that a *context-free grammar* $G=(N,T,P,S)$ consists of a set N of nonterminals, a set T of terminals, a set P of production rules, and a start symbol $S\in N$. A *parenthesis grammar* is a context-free grammar where "(" and ")" are distinguished terminals and every rule is of the form $A\rightarrow(x)$ with x parenthesis-free. A parenthesis language is a language generated by a parenthesis grammar.

Theorem 4.1: [Lyn77] *All parenthesis languages are recognizable in LOGSPACE.*

Lemma 4.2: *For every fixed database B, there exists a parenthesis language $L(B)$ such that $Answer_{FO^k}(B)$ is logspace-reducible to L.*

Proof sketch: The key observation is that for a fixed database B with domain D there is a fixed number of k-ary relations over D. Let these relations be $r_1,\ldots,r_l$. We now define a parenthesis grammar G with $S,r_1,\ldots,r_l$ as non-terminals and $\overline{r_1},\ldots,\overline{r_l}$ as terminals such that S derives $(\varphi@\overline{r_i})$ precisely when $(x_1,\ldots,x_k)\varphi(B)=r_i$. The terminal symbols of the grammar consist of:

- $\overline{r_1},\ldots,\overline{r_l}$,
- all atomic formulas of the form $Pu_1\ldots u_m$, where $u_1\ldots u_m$ are among $x_1,\ldots,x_k$ (since B has a fixed set of relations, there is a fixed number of such atomic formulas),
- $\neg$, @, $\wedge$, (,), and
- $\exists x_i$, for $i=1,\ldots,k$.

The grammar consists of the following productions:

- $S\rightarrow(r_i@\overline{r_i})$,
- $r_i\rightarrow(Px_{i_1}\ldots x_{i_m})$ if $r_i=(x_1,\ldots,x_k)Px_{i_1}\ldots x_{i_m}(B)$,
- $r_i\rightarrow(r_j\wedge r_k)$ if $r_i=r_j\cap r_k$,
- $r_i\rightarrow(\neg r_j)$ if $r_i=D^k-r_j$,
- $r_i\rightarrow(\exists x_j\, r_k)$, if r_i is obtained from r_k by projecting away the jth column.

It is not hard to see that $(\langle a_1,\ldots,a_l\rangle,(x_{i_1},\ldots,x_{i_l})\varphi)\in Answer_{FO^k}(B)$ precisely when $(\varphi@\overline{r_i})$ in in $L(G)$ for some tuple $\langle b_1,\ldots,b_k\rangle$ and relation r_i such that $\langle b_1,\ldots,b_k\rangle\in r_i$ and $a_j=b_{i_j}$ for $j=1,\ldots,l$. ∎

It follows from Theorem 4.1 and Lemma 4.2 that, for every fixed database B, $Answer_{FO^k}(B)$ is in LOGSPACE. We can actually get a sharper result. In [Bus87] it was shown that parenthesis languages are actually recognizable in ALOGTIME ($\subseteq$ LOGSPACE). The reduction of $Answer_{FO^k}(B)$ to the parenthesis language $L(G)$ in Lemma 4.2 involves guessing a tuple $\langle b_1,\ldots,b_k\rangle$ and a relation r_i whose size is *bounded*. Thus, we can check for membership in $Answer_{FO^k}(B)$ by asking a *fixed* number of $L(G)$-membership queries. It follows that:

Corollary 4.3: *For every fixed database B, $Answer_{FO^k}(B)$ is in ALOGTIME.*

Language	*Data Complexity*	*Expression Complexity*	*Combined Complexity*
FO	AC^0	PSPACE-complete	PSPACE-complete
FP	PTIME-complete	EXPTIME-complete	EXPTIME-complete
ESO	NP-complete	NEXPTIME-complete	NEXPTIME-complete
PFP	PSPACE-complete	EXPSPACE-complete	EXPSPACE-complete

Table 1: Complexity of Query Evaluation

$\mathcal{L}$	*Data Complexity of* $\mathcal{L}$	*Combined Complexity of* $\mathcal{L}^k$
FO	AC^0	PTIME-complete
FP	PTIME-complete	NP$\cap$co-NP
ESO	NP-complete	NP-complete
PFP	PSPACE-complete	PSPACE-complete

Table 2: Combined Complexity of Bounded-Variable Queries

$\mathcal{L}$	*Combined Complexity of* $\mathcal{L}^k$	*Expression Complexity of* $\mathcal{L}^k$
FO	PTIME-complete	ALOGTIME
FP	NP$\cap$co-NP	NP$\cap$co-NP
ESO	NP-complete	NP-complete
PFP	PSPACE-complete	PSPACE-complete

Table 3: Expression Complexity of Bounded-Variable Queries

In [Bus87] it is also shown that the *Boolean-value problem*, which by [Lyn77] is a special case of the parenthesis-language membership problem, is hard for ALOGTIME. It is easy to show that there is a database B such that the Boolean-value problem is reducible to $Answer_{FO^k}(B)$.

Theorem 4.4 : *There is a database B such that $Answer_{FO^k}(B)$ is ALOGTIME-hard.*

4.2 ESO^k and PFP^k

The NP-hardness and PSPACE-hardness lower bounds for the data complexity of ESO and PFP, respectively, were proven using a fixed query over varying databases. Here we prove the same bound using a fixed database and varying queries.

Theorem 4.5: *For every database B and $k \geq 0$, we have that $Answer_{ESO^k}(B)$ is NP-hard*

Proof sketch: Let φ be a propositional formula with propositions $P_1, \ldots, P_l$. Clearly, φ is satisfiable iff $\exists P_1 \cdots \exists P_l \varphi$ holds in B, regardless what B is. Thus, propositional satisfiability is reducible to $Answer_{ESO^k}(B)$. ∎

Theorem 4.6: *There exists a database B_0, such that $Answer_{PFP^k}(B_0)$ is PSPACE-hard.*

Proof sketch: The reduction is from QBF [GJ79]. The database B_0 consists of the domain $\{0,1\}$ and a unary relation $P = \{0\}$. For a given quantified Boolean formula

$$\forall Y_1 \exists Y_2 \cdots \forall Y_{l-1} \exists Y_l \theta(Y_1, \ldots, Y_l),$$

we write a PFP query φ that check whether the quantified Boolean formula holds. The query φ uses unary relation variables $X_1, \ldots, X_l$. The idea is that a relation variable X_i being empty or nonempty corresponds to the Boolean variable Y_i being false or true, respectively. By iterating through all possible assignments to the relation variables, the query simulates going through all truth assignments to the Boolean variables. ∎

5 Concluding Remarks

Our investigation of the complexity of bounded-variable query evaluation confirms the hypothesis that bounding the size of intermediate results eliminates the exponential gap between data complexity, on one hand, and expression complexity and combined complexity, on the other hand. The main question left open is the expression complexity and combined complexity of FP^k. Since the problem is in NP∩co-NP, one cannot hope to prove that the problem is NP-complete [GJ79], but so far the problem resisted all attempts to show that it is in PTIME.

Acknowledgement. I'd to thank Phokion Kolaitis for his helpful feedback on a previous draft of this paper. I am also grateful to the PODS program committee for their useful comments.

References

[Ajt83] M. Ajtai. Σ^1_1 formulae on finite structures. *Ann. of Pure and Applied Logic*, 24:1–48, 1983.

[And94] H.R. Andersen. Model checking and Boolean graphs. *Theoretical Computer Science*, 126(1):3–30, 1994.

[AV89] S. Abiteboul and V. Vianu. Fixpoint extensions of first-order logic and Datalog-like languages. In *Proc. 4th IEEE Symp. on Logic in Computer Science*, pages 71–79, 1989.

[BFMY83] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30:479–513, 1983.

[BIS90] D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.

[Bus87] S.R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 123–131, 1987.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, Stanford, California, June 1994. Lecture Notes in Computer Science, Springer-Verlag. full version available from authors.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[CH80] A. Chandra and D. Harel. Computable queries for relational databases. *Journal of Computer and System Sciences*, 21:156–178, 1980.

[CH82] A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25:99–128, 1982.

[Cha88] A. Chandra. Theory of database queries. In *Proc. 7th ACM Symp. on Principles of Database Systems*, pages 1–9, 1988.

[Cle90] R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27:725–747, 1990.

[Cle92] R. Cleaveland. Faster model checking for the modal mu-calculus. In *Proc. 4th Workshop on Computer Aided Verification*, volume 663 of *Lecture Notes in Computer Science*, pages 410–422, Montreal, 1992. Springer-Verlag.

[Cle93] R. Cleaveland. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th ACM Symp. on Theory of Computing*, pages 77–90, 1977.

[Cod72] E.F. Codd. Relational completeness of databases sublanguages. In R. Rustin, editor, *Data Base Systems*, pages 65–98. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[Coo74] S. A. Cook. An observation of time-storage trade-off. *Journal of Computer and System Sciences*, 9:308–316, 1974.

[Cos83] S.S. Cosmadakis. The complexity of evaluating relational queries. *Information and Control*, 58:101–112, 1983.

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, White Plains, October 1988.

[EJS93] E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, Proc. 5th Int. Workshop*, volume 697, pages 385–396, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the proposoitional mu-calculus. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 267–278, Cambridge, June 1986.

[Eme85] E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 79–87. Springer-Verlag, 1985.

[Fag74] R. Fagin. Generalized first–order spectra and polynomial–time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73, 1974.

[FSS81] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial time hierarchy. In *Proc. 22nd IEEE Symp. on Foundations of Computer Science*, pages 260–270, 1981.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[Hod93] I. Hodkinson. Finite variable logics. *Bulletin of the European Association for Theoretical Computer Science*, 51:111–140, October 1993.

[IK89] N. Immerman and D. Kozen. Definability with bounded number of bound variables. *Information and Computation*, 83:121–139, 1989.

[Imm82] N. Immerman. Upper and lower bounds for first-order expressibility. *Journal of Computer and System Sciences*, 25:76–98, 1982.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[Imm87] N. Immerman. Expressibility as a complexity measure: results and directions. In *Second Structure in Complexity Conference*, pages 194–202, 1987.

[Imm89] N. Immerman. Expressibility and paralel complexity. *SIAM Journal on Computing*, 18:625–638, 1989.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KV92] Ph. G. Kolaitis and M. Y. Vardi. Infinitary logic for computer science. In *Proc. 19th International Colloq. on Automata, Languages, and Programming*, pages 450–473. Springer-Verlag, Lecture Notes in Computer Science 623, 1992.

[Lei90] D. Leivant. Inductive definitions over finite structures. *Information and Computation*, 89:95–108, 1990.

[Lyn77] N. Lynch. Log space recognition and translation of parenthesis languages. *Journal of the ACM*, 24:583–590, 1977.

[SW91] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.

[Tar55] A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific J. of Mathematics*, 5:285–309, 1955.

[Ull89] J. D. Ullman. *Database and Knowledge-Base Systems, Volumes I and II*. Computer Science Press, 1989.

[Var82] M. Y. Vardi. The complexity of relational query languages. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 137–146, 1982.

[VS85] M.Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc 17th ACM Symp. on Theory of Computing*, pages 240–251, 1985.

[Win91] G. Winskel. Note on model checking the modal ν-calculus. *Theoretical Computer Science*, 83:157–167, 1991.

[Yan81] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. 7 Int'l Conf. on Very Large Data Bases*, pages 82–94, 1981.